UIUCDCS-R-73-555

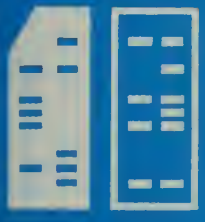
COO-1469-0215

NUMERICAL SYSTEMS ON A MINICOMPUTER

by

ROY LEONARD BROWN, JR.

February 1973



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-73-555

NUMERICAL SYSTEMS ON A MINICOMPUTER*

by

ROY LEONARD BROWN, JR.

February 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

* Supported in part by the Atomic Energy Commission under contract US AEC AT(11-1)1469 and submitted in partial fulfillment of the requirements of the Graduate College for the degree of Master of Science in Computer Science.

ACKNOWLEDGMENT

The assistance of the following people is greatly appreciated. My advisor, Professor C. W. Gear, provided both direction and cogent suggestions. Professors M. H. Pleck and R. L. Ruhl, and the Department of General Engineering provided the Illinois Graphics Computer System for my use in developing both the theory and the program presented here; and my colleagues at IGCS, W. F. W. Tam, D. Mueller, and T. Runge co-developed the software we all share. Mrs. Barbara Armstrong typed the manuscript. Finally, the Department of Computer Science and the U. S. Atomic Energy Commission supported the research.

PREFACE

This thesis defines the concept of a numerical system for a minicomputer and provides a description of the software and computer system configuration necessary to implement such a system. A procedure for creating a numerical system from a FORTRAN program is developed and an example is presented. The reader should have some knowledge of FORTRAN and minicomputer operating systems. A familiarity with PAL assembly language for the PDP-11 is necessary to make full use of the examples.

TABLE OF CONTENTS

1. THE PROBLEM

1.1 Definitions

A numerical system as used here is any long program involving large amounts of arithmetic computation on a set of numerical data that is small enough that manipulation of the data is not a significant part of the problem. Manipulation means movement of the data from one storage device to another or change from one format to another, and similar programming problems not ordinarily associated with numerical analysis. Some numerical systems would be, for example, programs designed to solve difference equations, find roots of nonlinear equations, perform numerical quadrature of an input function, minimize a function of several variables, or, in the example dealt with here, to solve a system of ordinary differential equations. All of these programs have in common the requirement of a great deal of arithmetic computation and simplicity of input: a number of polynomial coefficients, a difference equation of specified order and coefficients, a subroutine to evaluate a function with some initial conditions.

1.2 Minicomputer Use

Such a numerical system is often difficult to write for a minicomputer for several reasons. Most minicomputers have very little computing power for the vast amount of arithmetic operations needed to run a program of this type; some do not even have a hardware multiply/divide device, making complicated arithmetic operations time consuming. Small word sizes of 12 or 16 bits require that any floating point calculations use multi-word manipulations that take much time and storage

space. The small size of main memory means that the program will probably not all fit into it at one time. This also limits the amount of numerical data since storing it anywhere but main memory would be impossibly time consuming, given the block structure and slow access time of most peripheral storage devices.

In spite of these drawbacks, there are still several reasons for creating a numerical system for a minicomputer. Although running a numerical system is time consuming, it is not as expensive as the same amount of time on a System/360 computer. Some minicomputers are idle part of the day and a set of numerical systems, possibly in a program library for the installation, would fill in the gaps by being run in free time or at scheduled times daily. Making such programs available could reduce an organization's expenses if these programs are normally taken to a large service computer. The complexity of multi-word manipulation, transcendental function evaluation, and programming of complicated formulae is reduced greatly because many 16-bit word size minicomputer software packages include both a library of subroutines that perform multi-word manipulations, and a FORTRAN IV compiler that allows execution of FORTRAN programs with approximately the same accuracy as a System/360 computer using single or double precision arithmetic.

Finally, the problem of program size can be solved by proper use of peripheral storage devices such as a fixed head disc, drum, or other "random" access device. By storing large program segments that are always executed together on such a device (hereafter called disc although any "random" access device may be substituted), the program can be executed by bringing these segments into a core buffer, overlaying the program segment previously there. This core buffer will be called the overlay buffer.

1.3 Outline

This paper will deal primarily with a procedure for transforming a large FORTRAN numerical program known to work properly in a large computer environment into a numerical system that runs efficiently in a minicomputer environment. As a helpful example, a problem which has been solved using this procedure will be discussed wherever applicable. The coding of the example numerical system is found in Appendix B while the original problem is due to Gear [3] and involves integrating a first order system of ordinary differential equations through some interval of the independent variable, given a set of initial conditions and a subroutine to accomplish the differentiation. The original program, DIFSUB, is designed to handle stiff systems of equations as well as better conditioned ones.

The topics to be presented are listed below. First, a computer system which is well adapted to the problem is described; criteria for choosing similar systems are listed as well as references to several such systems. Next, the concept of a program overlay is developed. The application of this concept in partitioning a FORTRAN program will be presented, and then several problems resulting from such a partition will be discussed. These last will fully describe the procedure and some additional comments about it will conclude the paper.

2. THE COMPUTER SYSTEM

2.1 IGCS Hardware and Software

The Illinois Graphics Computing System used in the development of DIF11, the numerical system described in Section 1.3, belongs to the Department of General Engineering at the University of Illinois at Urbana-Champaign. As described in [5] and [6], it is based on a PDP-11/20 minicomputer from Digital Equipment Corporation (DEC). It has a byte addressable core memory of 16K 16-bit words with a ROM bootstrap loader. All communication between the CPU and peripheral devices takes place over a single bus which DEC calls UNIBUS. Access time for main memory is 490 ns. out and 8 ns. in. Instructions are basically of the two-address type, but 12 different addressing modes, eight general-purpose registers, and a software stack simulator allow a wide variety of instructions. Instruction formats require one to three words, depending on the addressing mode of operands. A KE11-A extended arithmetic unit provides single word hardware multiply and divide operations.

A 256K word fixed head disc, accessible through the available software in 64 word blocks, has an average access time of 17 ms. and a transfer rate of 16 μs/word. A card reader, teletype, and Gould 4800 electrostatic printer/plotter complete the basic system. The Gould 4800 is used by DIF11 as a printer through a software program, LP, since the Gould will eventually be the output device of a system containing both a CalComp-type graphical language and a simulation system of which DIF11 will be part [5].

The software package available with a minicomputer is very important since it can greatly reduce the work needed in programming a large system.

IGCS currently operates with a Disc Operating System, DOS version 4A, which includes utilities for transferring data between memory and peripheral devices, an assembler, a FORTRAN IV compiler, and a relocating Linkage Editor which presently includes an overlay facility. See [2] for details. EAELIB is a library of FORTRAN functions (e.g. SIN, COS) and utility routines (e.g. floating point addition) designed to operate with the KE11-A multiply/divide unit. These routines are linked with a FORTRAN program by the Linkage Editor and are used by the program at execution time.

This system is suitable for a numerical system since 16K of core is adequate for keeping the operating system core (DOS is not entirely resident, but overlays part of itself depending on its current operation), a set of routines from EAELIB for a large program, a main program, the overlay buffer(s), and storage for data whose size is variable and dependent on an input parameter. The FORTRAN compiler is very efficient in compiling code to do the actual computation involved, although control transfer, allocation of memory space, etc., are best performed using assembly language interfaces. FORTRAN programs interpret a polish string of addresses of routines placed in core by the Linkage Editor from EAELIB; these strings are the result of compilation of the program. Storage is best handled as FORTRAN arrays whose elements fill consecutive four byte segments of core and are easily processed by either FORTRAN or the assembly language interfaces. The original working version of DIF11 was implemented in 12K core with user-written overlay software so the present system is more than adequate. The speed of access, relatively small blocksize, and adequate device driver software make the fixed head disc easily accessible for transferring unchanging program segments to core.

2.2 Alternate Systems

Inspection of the above system leads to a set of criteria which a minicomputer system should satisfy to create a well suited environment for a numerical system. These criteria follow.

The main memory should be an adequate size to contain the system's resident monitor, a large section of code from the numerical system, and all the variable storage for the system. An adequate word size to minimize multi-word floating point manipulation is important; at least 16 bits is recommended. The execution speed should be fast enough to allow an adquate turnaround time for the size problem being run; a good criterion would be that the minicomputer turnaround time including loading the numerical system and execution should be no more than 10 times the turnaround for a full size service computer from input to retrieval of output.

The peripheral device that acts as a bulk memory, usually a fixed head disc or drum, should have fast transfer into core. Any sequential device would be unsuitable for this.

An efficient FORTRAN compiler (i.e., programs compiled use close to the absolute minimum of core) is necessary. Since many numerical systems may need to use double precision arithmetic, this feature should compile and execute efficiently.

Finally, a good software monitor capable of fast and effective control of device transfers while occupying a minimum amount of main memory is important, both at execution time and during creation of the numerical system.

Several computer systems, while dissimilar to the PDP-11/20 in many ways, appear to meet the above criteria. Some may be better

suited to the purposes of the purchaser than others and still provide an acceptable environment for numerical systems. They are:

Hewlett-Packard 2100 A [4]

Interdata Model 70 [7]

Systems 72 [8]

Varian 620/f [9]

Westinghouse 2500 [10]

3. THE OVERLAY CONCEPT

3.1 Definitions

A program overlay is a segment of executable code intended to be used with a permanently resident program segment. However, this code is stored permanently only on a peripheral storage device (disc) and is brought into a core buffer only when needed. When another overlay is needed instead, that overlay is brought from disc and placed into the same buffer, thus "overlaying" the previous segment. Clearly, N overlays of approximately equal size effectively expand the overlay buffer to simulate a memory area N times as large. However, one pays for this in terms of the cost of disc to core transfer time. This is more advantageous than having all executable code in core (if possible) and using the disc to store variables because these must be transferred not only into core, but also back from core to disc after being processed, whereas overlays are constantly "refreshed" from an unchanging disc image.

Since overlays for numerical systems are usually parts of a FORTRAN program originally intended to be executed in the same main memory, the overlay should have access to all variables stored in resident core, and to all resident subroutines, especially such utility routines as are normally used by minicomputer FORTRAN compilers for complicated arithmetic operations such as floating point addition (e.g. EAELIB in Section 2.1). Because of these features, certain control paths between the resident code and each overlay must be provided. These paths are:

a. Resident to overlay

b. Overlay to overlay

c. Overlay to resident; return of control

d. Overlay using resident subroutines

A core map of an idealized numerical system is in Figure 3.1.1.

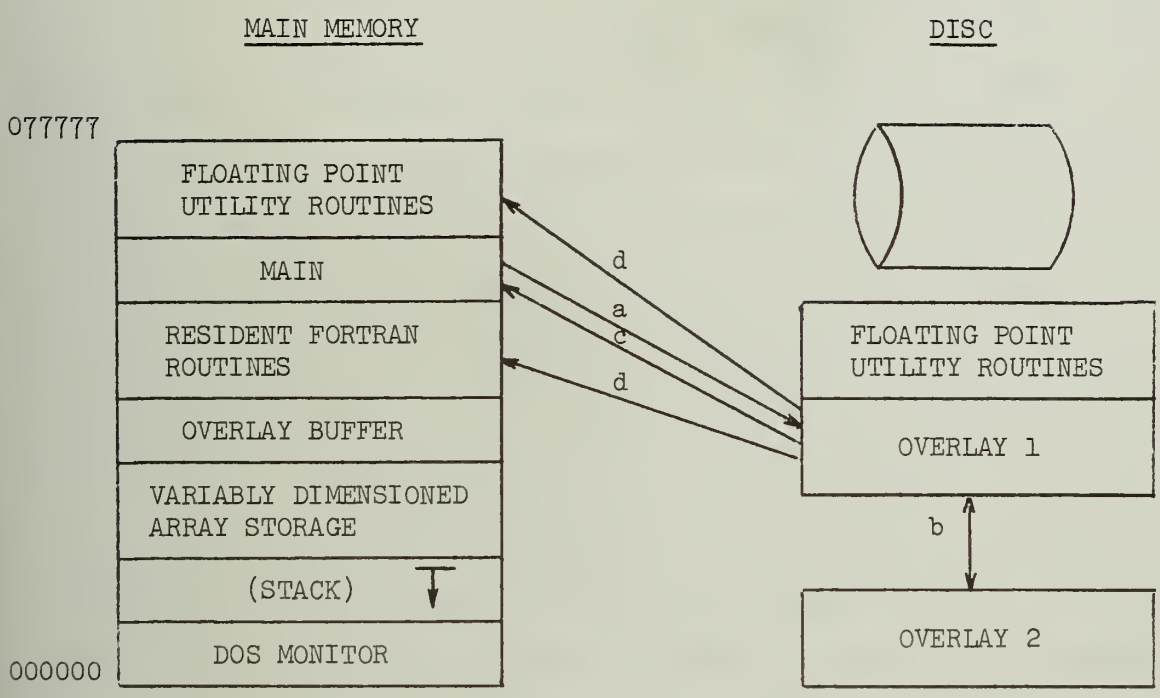


Figure 3.1.1. Sample Program Map with Control Paths Indicated

3.2 User-written Overlay Facilities

Some minicomputer software systems do not have overlay facilities; in these cases the user must code his own. Such a system was written for use with IGCS before versions of FORTRAN and the Linkage Editor were provided with one. The code for this system in PAL assembly language is in Appendix A, and a core image for creation of an overlay file with it is in Figure 3.2.1.

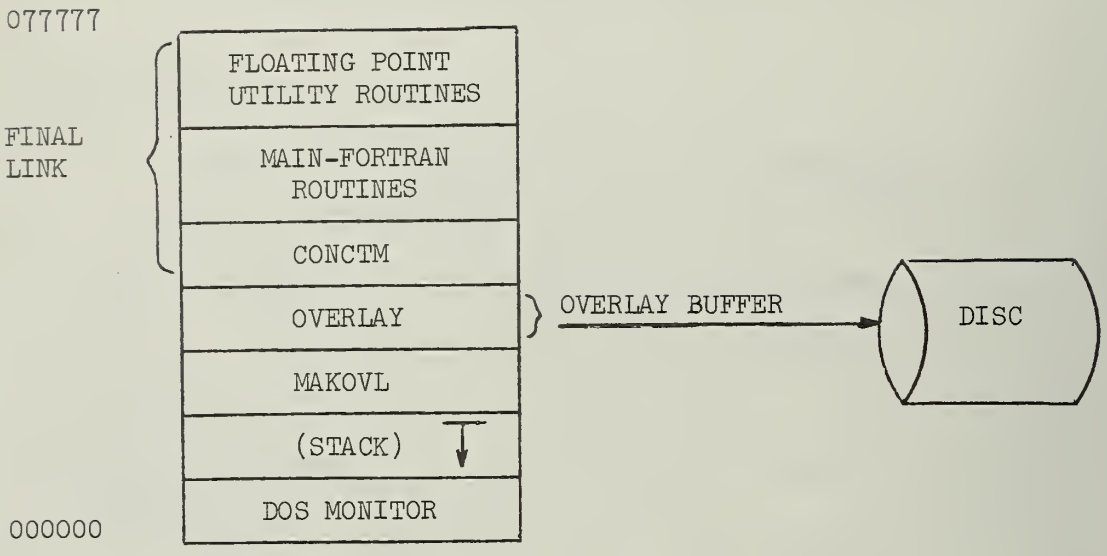


Figure 3.2.1. Core Map for Creating an Overlay Using MAKOVL

The system consists of two programs called CONCTM and MAKOVL. CONCTM is a core resident subroutine which contains tables of the file names of all overlays, their length in words, and their starting addresses in core. On being called, it looks up the disc start block of the overlay using the FILE BLOCK and LINK BLOCK [2] and places these in the TRAN BLOCK. The overlays are contiguous on disc, i.e., if they occupy more than one disc block, they are placed in consecutive disc blocks. When an overlay is called, the call is to CONCTM with the first argument the integer designating the overlay wanted and the following arguments being the proper arguments for the subroutine from which the overlay was made. CONCTM transfers the overlay into core at the indicated starting address, using a TRAN request and the TRAN BLOCK mentioned above. Since the addresses of the arguments of the overlay subroutine follow the statement JSR CONCTM,R5 in the main program, and since subroutine arguments are referenced relative to the return address stored in register R5, a simple transfer to the start

address of the overlay starts its execution. A FORTRAN RETURN statement shifts control back to the resident program segment.

The overlays are created by linking together all resident parts of the program, the overlay, and MAKOVL below these (the Linkage Editor links in the order of its input string, filling the top of core). MAKOVL is executed from the keyboard and computes the length in words and disc blocks of the overlay and transfers the overlay to disc under the name placed in its FILE BLOCK (this can be set from the keyboard). The user saves the length and starting address and inserts them into the final version of CONCTM when the routines indicated as the final link in Figure 3.2.1 are linked together as the initial load module.

Here it is difficult to separate the floating point package routines between those needed by the resident segment and those needed only by the overlays. One solution is to determine all utility routines needed by any program and force them to be permanently resident. On the PDP-11/20, this can be done by linking each program and inspecting its load map which lists all utility routines called by it. The union of all utility routines listed on any load map is then placed in a GLOBL statement. A version of DIF11 using the above system was programmed and tested.

3.3 Manufacturer-supplied Overlay Facility

The overlay facility described here is found in more detail in [1]. It is typical of those available from manufacturers' software groups. The system uses a single multiply entrant subroutine LINK with entry points LINK and RETURN. These provide all the control paths described in Section 3.1. CALL LINK('FILE') in resident FORTRAN code or JSR LINK,R5 with

the proper argument in assembly language 1) initializes FORTRAN traceback routines, 2) saves register 0-5, 3) causes the named file to be found, transferred into core, and executed as if it were a main program. The statement CALL RETURN restores register 0-5 and returns control to the place in resident code where LINK was last called.

Since it is easiest to write and test the FORTRAN part of the numerical system with the overlays as subroutines, a problem arises in that the overlays must be set up as main program segments. In some cases the FORTRAN SUBROUTINE statement can be removed, but if any parameters must be passed as if the overlay were a subroutine (see Section 5.2), then an assembly language main program can be written with the appropriate arguments. Such an example is the routine INTRO in Appendix B.

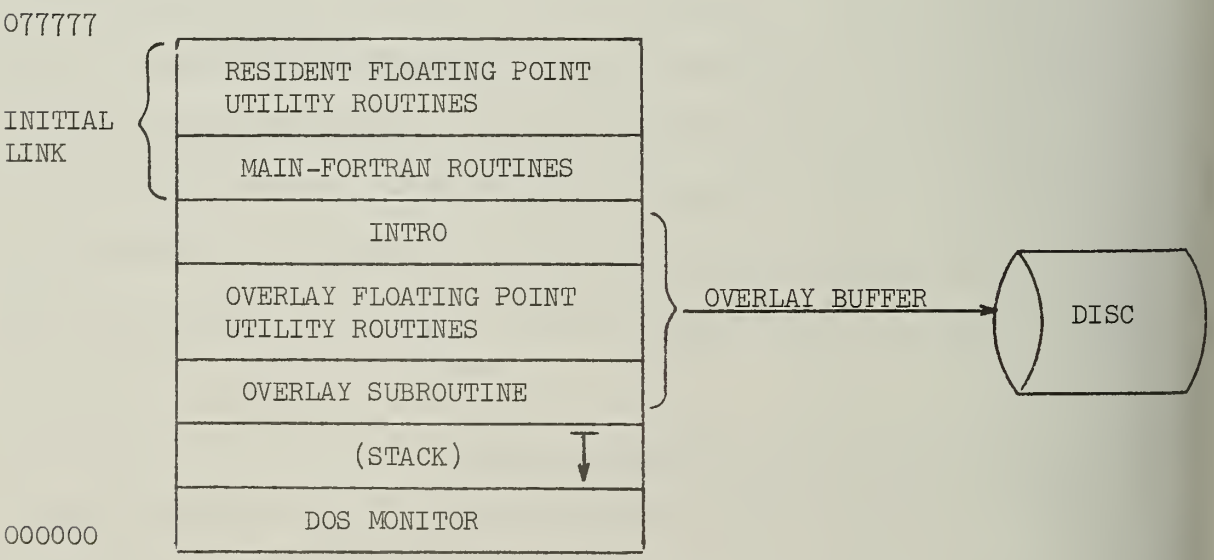


Figure 3.3.1. Core Map for Creating an Overlay Using Manufacturer-supplied Software

The overlays are created using the relocating Linkage Editor, LINK. The resident program segments are first linked together, bringing

all floating point utilities used by the resident code into core. Two of the outputs of LINK are saved: the object module, and the Symbol Table which lists all subroutine start addresses including those for the utility routines. Then each overlay (with its main program INTRO if needed) is created; the first input to LINK is the Symbol Table, and the top address of the output load module is set to two less than the bottom of the core resident module just created. The Symbol Table input allows the overlay to use utility routines already core resident and only utility routines which it alone needs are added to the overlay. Figure 3.3.1 shows the configurations of these different uses of LINK to create the resident and overlay modules.

4. PARTITIONING THE FORTRAN PROGRAM

4.1 Criteria for Partitioning

After providing for an overlay facility, one next considers the overlays themselves. It is best to begin with a FORTRAN program which is known to run in a large computer environment and insure that the overlay interconnections will work by breaking the program down into overlay subroutines and running it again on the same computer.

There are really no hard and fast rules that can be applied, but since we are dealing with relatively slow minicomputers and one of the slower operations is disc-to-core transfers, one goal of the partitioning process is to minimize the number of such transfers.

Since core space is at a premium, most of the original program should be put into overlays, leaving minor bookkeeping arithmetic and major control transfer decisions in the resident program segment; by making the overlays of nearly uniform length, the overlay buffer (which is as long as the longest overlay) is also made as small as possible. The implementations of these two goals are clearly at odds since the real minimum number of overlay transfers from disk would maximize the overlay buffer, and the minimum overlay size would be only two or three FORTRAN statements, requiring innumerable disc transfers, usually of length less than one disc block. Proper implementation calls for a balance between these two extremes.

4.2 COMMON Variables

Because the contents of the overlay buffer at the time a new overlay is transferred in will be overwritten, any FORTRAN variables

(other than special constants) must be core resident. This is most easily accomplished with COMMON statements which, when linked into load modules by most Linkage Editor facilities, leave all of the variables in a resident core block. When the overlay modules are linked, the addresses of elements in the COMMON block are available from the Symbol Table and are linked with the overlay. To form the COMMON block, each overlay and the resident program is inspected and all of its fixed length variables are listed. A variable array is of fixed length if its dimension does not depend on an input to the program. Next, any variable appearing on any two lists must obviously be shared by two different routines and is placed in the COMMON block. Any data that remains the same in an overlay and is not shared by any other overlay can be defined by a DATA statement and need not be in the COMMON block. It may be convenient to have several COMMON blocks since some variables may be used exclusively by certain overlays and no others. Allocation of space for arrays with variable dimensions will be discussed in Section 5.3, but note that the initial address of each such variable must be passed to the overlay as a subroutine parameter (it cannot be in a COMMON block since its initial address is not available at Linkage time), and the variable dimension must be passed to the overlay as a subroutine parameter as required by most FORTRAN compilers.

4.3 Program Logic

Some few FORTRAN programs are executed sequentially, i.e., starting at the first statement and "dropping through" to the end with a few tight DO-loops. Partitioning these is trivial. The program is divided into equal

length overlay segments; either each overlay calls the next, or, if this is not possible, a short resident program calls each overlay in sequence. However, most programs are not this easy.

All or part of the more usual program is executed iteratively either a fixed number of times or until certain conditions are met. DIF11, for example, predicts next values of the differential variables, corrects these values, and, under certain conditions, changes other control variables (order, step size, etc.). This whole cycle--predict, correct, change--is repeated until a final value is reached for the independent variable.

The best approach to partitioning such a program is to follow a principle of least use; if something is seldom used, it is an ideal candidate for an overlay. Many programs have a long initialization section which is only executed the first time the program is used. Clearly, this should be an overlay. The original version of DIF11 had such an overlay but the present version does not need one since the initialization was shortened. A segment that is executed at most once per iteration is the logical next candidate. In DIF11, SUB3 and SUB5 are executed only once per iteration and SUB1, SUB2, and SUB6 are executed only when a change of control variables is being considered or carried out. (See Appendix B).

DO-loops are best placed completely within an overlay to avoid each pass through the loop causing one or more disc transfers to the Overlay buffer. Occasionally, however, a DO-loop will contain enough material for several overlays (remembering that uniform overlay size is a goal of the partition), and the DO-loop then contains several overlays.

DIFFUN, SUB7, and SUB4 are all contained in one loop in DIF11. The same loop contains code for the numerical evaluation of the Jacobian matrix which could have been an overlay. It was not because this would have required one overlay to call another and to return to the original overlay. This complicated process is best avoided but could be handled by using two overlay buffers. See Section 5.2.

Occasionally, one especially long overlay may be needed but only part of it need be in core the whole time it is being executed. In such a case only enough of the overlay should be brought into core at one time to fill the regular sized overlay buffer, and when some part of it that will not be executed again is completed, the rest is overlayed there. This will be possible if the overlay facility allows overlays to call overlays as specified in Section 3.1.

Any segment that is called from a number of different places in the resident code is probably best left in core, but if it is long or called very seldom, it can be made an overlay as in SUB8 in DIF11.

4.4 Control Transfer

It must be noted that each overlay may have several GO TO statements which reference a statement in the resident module, or worse, in another overlay. The partition should minimize this, but if it happens, than a control transfer vector can be used to allow this. Either a single variable, JLINK, or an array, JLNK(N), can be used; the various values should have some significance to the program. The single variable is probably better since it is less complicated. Set JLNK = 1 on entering

an overlay, then if a statement is, say

```
GO TO 750
```

and 750 is in the resident module, change the statement to

```
JLNK = 2
RETURN
```

If a statement is

```
IF(LOGICAL EXPRESSION)GO TO 715
```

change this to

```
IF(LOGICAL EXPRESSION)JLNK=3
IF(JLNK.EQ.3)RETURN
```

This requires that any arithmetic IF statements be made logical IF statements.

Immediately after the overlay calling statement in the resident code, a computed GO TO statement can be used to effect the transfer:

```
      CALL OVERLAY (SUB5)
      GO TO(540,750,715,...),JLNK
540   ...
```

remembering the possibility of executing the next sequential statement if JLNK remains 1.

Transfers into the middle of another overlay can be handled by a computed GO TO at the beginning of the overlay so that this is the first statement other than subroutine calls after the computed GO TO in the resident code. JLNK must be set to 1 after executing the initial computed GO TO statement. For example,

```
                GO TO 700
```

becomes

```
                JLNK=2                          (overlay 1)
                RETURN
          * * *  * * * * * * * * * *
                 :
           540   GO TO(550,600),JLNK             (MAIN)
                 :
           600   CALL OVERLAY(SUB2)
          * * *  * * * * * * * * * *
                 :
                 GO TO(650,700),JLNK
                 :                              (overlay 2)
           700   JLNK=1
                 :
```

Examples of all these techniques are in SUB5 of DIF11 in Appendix B.

5. USER SUPPLIED OVERLAYS

5.1 User Input

Many numerical systems can operate with a minimum of input. For example, the only input needed to solve a difference equation is the order, the coefficients, and the initial values. These can all be input on cards or keyboard, and the storage for this data can be in the core resident main program by specifying a maximum size for each variable array. Other programs will require more from the user than numerical data; a numerical quadrature program will require that some way be provided to evaluate the function at any given point, usually a subroutine. Some programs will have large data arrays which differ greatly in size depending on input parameters. Storing these within the main program segment may be too restrictive. An LU matrix decomposition would need such a variable sized space to store the matrix and outputs. And, some systems require both an evaluating subroutine and variable sized arrays, as is the case with DIF11.

5.2 User Subroutines

The solutions to these problems for DIF11 are presented here; they seem applicable to similar cases on other systems and are at least a good starting point. In DIF11, the user is responsible for writing a FORTRAN subroutine called DIFFUN(T,Y,DY), and by following carefully detailed instructions, creating an overlay from it. First, compile DIFFUN(T,Y,DY) where T is the independent variable, DY(I) is the first derivative of Y(I), and all derivatives of order m higher than one have been replaced by m first-order equations using standard techniques.

Y(I) and DY(I) should be dimensioned N, the number of equations. Next, call LINK, the Linkage Editor. Specify DIFFUN.LDA as output; specify as input ST (the Symbol Table of Section 3.3 which is kept available on disc or tape), the object module of INTRO (also available), the object module resulting from compiling DIFFUN, and the EAELIB. Also, specify that the top core address available to this module is two less than the bottom of the resident code (this address is posted in the machine room). After running LINK, run the program DIF11 with two data cards: one specifies N, T_0, H (the length of the integration), EPS (the error criterion), and ISTORE (the octal top address to be filled by dynamically allocated variable arrays); the other lists the N initial values of Y(I).

This overlay uses the same overlay buffer as all other overlays. This is possible because none of the other overlays call it. Any other overlay could call DIFFUN and then return to the main program; however, a great deal more work would be required to let an overlay call DIFFUN, have DIFFUN reinstate that overlay, and then transfer to the statement after the call to DIFFUN. This would be equivalent to a regular subroutine call to DIFFUN.

Another possibility is to provide a second buffer just for the user supplied overlay, allowing all overlays to call it. However, it must be remembered that this takes valuable core that might otherwise be used for variable storage. Since there is no way to predict the length of a user supplied module, this allows the first overlay buffer to be of fixed length while the second varies. This may be an important consideration for some users.

5.3 Variable Size Arrays

The problem of variables whose dimensions depend on an input parameter is best solved by allocating storage space for the FORTRAN arrays in an assembly language MAIN program. In DIF11, various arrays are dimensioned (N,7), (N,2), or (N,N). The MAIN program (see Appendix B) computes 4*N (each floating point word uses four bytes on the PDP-11/20), 28*N, 8*N, and 4*N**2. Using R0 (register 0) initially set to ISTORE as a pointer, MAIN subtracts an appropriate number of bytes from R0 and assigns R0 to an address vector ADDR. Then it places the addresses of the appropriate variables in the address list of the calling sequences of INPUT, OUTPUT, and DIFSUB--the FORTRAN subroutines called from MAIN--and executes. It should be noted that most FORTRAN compilers require a variably dimensioned array, e.g. Y(N,7), to be in a subroutine with the variable dimension N as an argument, so DIFSUB, INPUT, and the overlays must be subroutines. Whenever DIFSUB calls one of its overlays, the overlay starts with a main program INTRO as required by the overlay facility, and INTRO places the initial addresses assigned to the variably dimensioned arrays and kept in an assembly language named .CSECT block (same as FORTRAN COMMON) into the FORTRAN calling sequence. Then INTRO executes the subroutine call, returning to the overlay main program before calling RETURN (see Section 3.3) in order to restore all registers. In this way, eight variably dimensioned arrays are allocated dynamically in a manner similar to ALGOL by using assembly language interfaces between FORTRAN program segments.

It should be noted that the pseudo-stack mentioned in Section 2.1 is kept immediately below the overlay buffer by the DEC provided software,

and extends into lower core addresses. This requires that the dynamically allocated variables be placed well below the stack and is the reason ISTORE is an input parameter. If a user written overlay facility were available, the stack could be moved at will below the data.

6. CONCLUDING OBSERVATIONS

The following observations seem appropriate to the problem discussed and are presented in an attempt to provide some perspective. Firstly, the method of reducing a large FORTRAN program to a minicomputer overlay system is best considered as a procedure. A list of step by step instructions can be found in the preceding chapters, but no guarantee is provided that the procedure will ever stop. Thus, this is not an algorithm with input: one large FORTRAN program; output: equivalent minicomputer numerical system. A counter example is a program with larger storage requirements than could ever fit into core or be profitably allocated to disc. This procedure can be helpful but no promises are made.

Secondly, in most cases the more general a program is, the larger it becomes. A simple Runge-Kutta (R-K) program with step doubling will take much longer to integrate a complicated system on a 360/75 than DIFSUB will; but if a problem is fairly simple, a R-K program requiring no overlays would be much faster than DIF11 on the PDP-11/20. Thus, one can save time by having available a simple method for simple problems as well as a complicated method to solve complicated problems.

Finally, one should be wary of manufacturer supplied overlay facilities, especially if overlays are called more than a few times. The overlay processor supplied by DEC looks up the current disc start block of each overlay every time it is called, even though some overlays are called hundreds of times from the same disc location. For this reason, the final simulation language package, ILLISM, which DIF11 will be part of, will use a locally written overlay facility that looks up disc addresses once, stores them in a directory, and thus executes twice as fast.

With these comments in mind, one can conclude that many large numerical analysis programs can be profitably used on a minicomputer system using the techniques described here.

LIST OF REFERENCES

[1] Digital Equipment Corporation, "Getting FORTRAN on the Air," (DEC-11-SFDC-D), Maynard, Massachusetts, 1972.

[2] Digital Equipment Corporation, "DOS Monitor Programs Handbook," (DEC-11-SERA-D), Maynard, Massachusetts, 1971.

[3] Gear, C. W. NUMERICAL INITIAL VALUE PROBLEMS IN ORDINARY DIFFERENTIAL EQUATIONS, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

[4] Hewlett-Packard Company, "2100A Computer Reference Manual," (102100-90001), Cupertino, California, 1970.

[5] Pleck, M. and Ruhl, R., "Illinois Graphics Computing System," Progress Report #1, Department of General Engineering, University of Illinois at Urbana-Champaign, 1971.

[6] _____, Progress Report #2.

[7] Interdata, Inc., "Model 70 User's Manual," (29-261), Oceanport, New Jersey, 1971.

[8] Multidata, Inc., "Computer Reference Manual--Systems 72," (31101), Westminster, California, 1970.

[9] Varian Data Machines, "Varian 620/f Computer Handbook," (98-A 9908 001), Irvine, California, 1970.

[10] Westinghouse Computer and Instrument Division, "Computer Reference Manual," (25 REF-001), Orlando, Florida, 1971.

APPENDIX A

User-written Overlay Facility

```
          .TITLE  MAKOVL
          .CSECT.
          .GLOBL DIFFUN,CONCTM
SP = %6
MAKOVL:    MOV #CONCTM,OVRLEN
          SUB #DIFFUN,OVRLEN
          MOV   #LNKBLK,-(SP)
          EMT  6
;GET NUMBER OF 64 WORD SEGMENTS  AND NUMBER OF WORDS.
          MOV  OVRLEN,NUMSEG
          ASR  NUMSEG
          ASR  NUMSEG
          ASR  NUMSEG
          ASR  NUMSEG
          ASR  NUMSEG
          ASR  NUMSEG
          ASR  NUMSEG
          ADD  #1,NUMSEG
;         ALLOC  LNKBLK,FILBLK,NUMSEG
          MOV   NUMSEG,-(SP)
      MOV   #FILBLK,-(SP)
          MOV   #LNKBLK,-(SP)
          EMT 15
          INC  (SP)+
          BNE  ERR
;         .OPENC  LNKBLK,FILBLK
          MOV  #13,FILBLK-2
          MOV #FILBLK,-(SP)
          MOV  #LNKBLK,-(SP)
          EMT 16
          MOV  #BUF,-(SP)
          MOV #LNKBLK,-(SP)
            EMT 2
;        WAIT  LNKBLK
          MOV  #LNKBLK,-(SP)
          EMT 1
;         .CLOSE   LNKBLK
          MOV  #LNKBLK,-(SP)
          EMT 17
;         .RLSE  LNKBLK
          MOV  #LNKBLK,-(SP)
          EMT 7
;         .EXIT
ERR:      EMT 60
OVRLEN:    .WORD  0
NUMSEG:   .WORD 0
;       LNNK BLOCK
          .WORD 0
LNKBLK:   .WORD 0,0,1
          .RAD50  /DF/
;    FILE BLOCK
          .WORD 0,0
FILBLK:   .RAD50  /DIF/
          .RAD50  /FUN/
         .RAD50 /LDA/
          .BYTE  0,0,0,0
BUF:      .WORD  OVRLEN,7,OVRLEN,DIFFUN
          .END
```

```
;       SUBROUTINE CONCTM
          .TITLE  CONCTM
          SP = %6
          R0 = %0
          .GLOBL CONCTM
CONCTM:    MOV @2(%5),POINTR        ;MOVE SR INDICATOR TO POINTR
          ASL    POINTR
          ASL    POINTR             ;PUT 4*POINTR ON THE STACK.
          MOV  R0,-(SP)
          MOV    POINTR,R0       ;  PUT SECOND WORD OF MODULE 5A45
          MOV POINTR(R0),FILBLK+2  ;FILE BLOCK
          MOV    POINTR-2(R0),FILBLK  ; PUT FIRST WORD IN FILE BLOCK
          MOV    ADDR(R0),TRNBLK+4
          MOV    ADDR-2(R0),TRNBLK+2   ;PUT START LOCATION IN TRAN BLOCK
          MOV    (SP)+,R0
          MOV    #LNKBLK,-(SP)
          EMT    6               ;.INIT
          MOV    #FILBLK,-(SP)
          CLR    -(SP)
          MOV    #LNKBLK,-(SP)
          EMT 14                 ;       .LOOK
          MOV    (SP)+,TRNBLK     ;PUT START BLOCK IN TRAN BLOCK
          TST   (SP)+
          TST   (SP)+
          MOV    #TRNBLK,-(SP)
          MOV    #LNKBLK,-(SP)
          EMT    10              ;.WRITE
          MOV    #LNKBLK,-(SP)
          EMT    1               ;  .WAIT
          MOV    #LNKBLK,-(SP)
          EMT   7                ;  .RLSE
          JMP    @TRNBLK+2       ;  GO TO SUBROUTINE START ADDRESS
POINTR:    .WORD   0         ;POINTER AND LIST OF FILE NAMES
         .RAD50   /DIF/
         .RAD50   /FUN/
         .RAD50   /PAS/
         .RAD50   /CAL/
         .RAD50   /AFI/
         .RAD50   /ND/
         .RAD50   /SCA/
         .RAD50   /LE/
         .RAD50   /EVA/
         .RAD50   /L/
         .RAD50   /CHA/
         .RAD50   /NGE/
          .RAD50   /INI/
          .RAD50   /T/
          .RAD50   /COR/
          .RAD50   /ECT/
NUMWRD:ADDR:    .WORD 0,37260,76,37122,155,37112,161,36476,367
          .WORD   36634,310,35864,1174,37072,161,36414,420
          .WORD   0       ;LINK BLOCK
LNKBLK:    .WORD   0,0,1
          .RAD50    /DF/
          .WORD   0,4     ;FILE BLOCK
FILBLK:    .WORD   0,0
          .RAD50    /LDA/
          .BYTE    0,0,0,0
TRNBLK:    .WORD   0,0,0,4,0
          .END
```

APPENDIX B

DIF11 Program Code

```
;
;
;       THIS PROGRAM IS THE MAIN PROGRAM OF DIF11.
;       IT INITIALIZES CONSTANTS, READS IN VARIABLES,
;       AND ASSIGNS STORAGE LOCATIONS TO VARIABLY
;       DIMENSIONED STORAGE FOR DIFSUB.
;
;
.TITLE MAIN
R0 = %0
R1 = %1
R2 = %2
R3 = %3
R4 = %4
R5 = %5
SP = %6
PC = %7
.GLOBL LOW1,LOW2,OUTPUT,INPUT,LINK,RETURN,N,DIFSUB
.CSECT
BEGIN:    CLR N                 ;CALL INPUT SUBROUTINE WITH N=0 TO READ IN
        JSR R5,INPUT            ;N,T,H,EPS,AND LOW2=BOTTOM ADDRESS OF
          BR RET1               ;DIFFUN OVERLAY PROVIDED BY USER.
.WORD 0,N,N2,LOW2,XLIM
RET1:     CMP LOW1,LOW2         ;LOWEST OVERLAY START ADDRESS GOES TO LOW1
          BMI ON                ;TO BECOME TOP OF STORAGE AREA.
          MOV LOW2,LOW1
ON:       MOV N,R1              ;COMPUTE ADDRESS OF DYNAMICALLY ALLOCATED
          ASL R1                ;STORAGE FOR DIMENSIONED VARIABLES.
          ASL R1
          MOV LOW1,R0           ;R1 CONTAINS 4*N,R0CONTAINS TOP OF STORAGE.
          SUB #2,R0
          SUB R1,R0             ;ADY = # OF DY(N)
          MOV R0,ADY
          SUB R1,R0
          MOV R0,AERROR         ;AERROR = # OF ERROR(N)
            SUB R1,R0
          MOV R0,AYMAX          ;AYMAX = # OF YMAX(N)
          SUB R1,R0
          MOV R0,AIP            ;AIP = # OF IP(N)
          MOV R1,R3
          ASL R3
          ASL R3
          ASL R3
          SUB R1,R3             ; R3 CONTAINS 7*4*N
          SUB R3,R0
          MOV R0,AY             ;AY = # OF Y(N,7)
          SUB R3,R0
          MOV R0,ASAVE          ;ASAVE = # OF SAVE(N,7)
          ASL R1                ;R1 CONTAINS 8*N
          SUB R1,R0
          MOV R0,ACSAVE         ;ACSAVE = # OF CSAVE(N,2)
          SUB N2,R0             ;N2 = (N**2)*4
          MOV R0,APW            ;APW = # OF PW(N,N)
          BR .+4
.WORD 0
          MOV AY,LISTO
          JSR R5,INPUT          ; READ Y(*,I) FROM CARDS
          BR RET2
LISTO:    .WORD 0,N,0,0,0
RET2:     MOV  AYMAX,R1         ;SET YMAX(*) = 1.0
          MOV N,R0
LOOP3:    MOV #40200,(R1)+
          CLR (R1)+
          SUB #1,R0
          CMP #0,R0             ;R0 IS THE LOOP COUNTER
          BNE LOOP3
          MOV AY,OUTLIS         ;PREPARE PARAMETER LISTS FOR CALLING
          MOV #20,R3
          MOV #ADDR,R0          ;DIFSUB AND OUTPUT
          MOV #LIST,R1
LOOP4:    MOV (R0)+,(R1)+
          TST (R3)-
          CMP #0,R3
          BNE LOOP4
          CLR JSTART            ;JSTART = 0
          BR ST20
ST19:     MOV #1,JSTART         ;JSTART = 1
ST20:     JSR R5,DIFSUB
          BR OUT
          .WORD N
LIST:     .WORD 0,0,0,0,0,0,0,0,0
OUT:      JSR R5,OUTPUT
          BR RET3
OUTLIS:   .WORD 0,N,XLIM
RET3:     CMP #1,KFLAG
          BEQ ST19
          EMT 60
```

```
N:          .WORD 0,0
N2:         .WORD 0,0
LOW1:       .WORD 033512,0
LOW2:       .WORD 0,0
XLIM:       .WORD 0,0
.CSECT COM
JSTART = .+40
KFLAG = .+50
T = .+74
H = .+124
HMAX = .+140
HMIN = .+144
.CSECT ADDR
ADDR:ADY:    .WORD 0
AERROR: .WORD 0
AYMAX:      .WORD 0
AIP:        .WORD 0
AY:         .WORD 0
ASAVE:      .WORD 0
ACSAVE:     .WORD 0
APW:        .WORD 0
.END BEGIN
```

Resident I/O Routines

```
      SUBROUTINE INPUT(Y,N,N2,LOW2,XLIM)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
      DIMENSION Y(N,7)
1000  FORMAT(I5,3G20.0,06)
2000  FORMAT(4G20.8)
      IF(N.GT.0) GO TO 10
      READ(5,1000) N,T,H,EPS,LOW2
      XLIM = T + H
      HMAX = H
      H = HMAX/1000.
      HMIN = H/1000.
      N2 = N*N*4
      RETURN
   10 READ(5,2000) (Y(I,1),I=1,N)
      RETURN
      END
```

```
      SUBROUTINE OUTPUT(Y,N,XLIM)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
      DIMENSION Y(N,7)
      WRITE(3,1001) T,(Y(I,1),I=1,N)
1001  FORMAT(5E16.7)
      IF(T.GT.XLIM) KFLAG=0
      RETURN
      END
```

```
      SUBROUTINE DIFSUB(N,DY,ERROR,YMAX,IP,Y,SAVE,CSAVE,PW)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,EN01,
     2EN02,EN03,BND,BR,DEL,DEL1,D
C*********************************************************************
C*                                                                   *
C* THIS SUBROUTINE INTEGRATES A SET OF N ORDINARY DIFFERENTIAL FIRST  *
C* ORDER EQUATIONS OVER ONE STEP OF LENGTH H AT EACH CALL.  H MAY BE  *
C* INCREASED OR DECREASED WITHIN THE RANGE HMIN TO HMAX TO            *
C* ACHIEVE AS LARGE A STEP AS POSSIBLE WHILE NOT COMMITTING A SINGLE  *
C* STEP ERROR WHICH IS LARGER THAN EPS IN THE L-2 NORM, WHERE EACH    *
C* COMPONENT OF THE ERROR IS DIVIDED BY THE COMPONENTS OF YMAX.       *
C*                                                                   *
C* THE PROGRAM REQUIRES THE  SUBROUTINES NAMED                       *
C*    DIFFUN(T,Y,DY)                                                 *
C*    DECOMP(N,M,PW,IP)                                              *
C*    SOLVE(N,M,PW,CSAVE(1,1),IP)                                    *
C* THE FIRST, DIFFUN, EVALUATES THE DERIVATIVES OF THE DEPENDENT      *
C* VARIABLES STORED IN Y(1,I) FOR I = 1 TO N, AND STORES THE          *
C* DERIVATIVES IN THE ARRAY DY. DECOMP IS A                          *
C* STANDARD LU DECOMPOSITION WITH PIVOTING THAT DECOMPOSES THE MATRIX *
C* PW, LEAVING THE PIVOTS IN THE INTEGER ARRAY IP.  M IS THE DECLARED *
C* SIZE OF PW.  IP(N) IS SET TO 0 IF PW IS SINGULAR.  SOLVE PERFORMS  *
C* BACK SUBSTITUTION ON THE CONTENTS OF CSAVE(I,1), LEAVING THE       *
C* RESULTS THERE.                                                    *
C*                                                                   *
C* THE TEMPORARY STORAGE SPACE IS PROVIDED BY   *MAIN*   IN THE      *
C* INTEGER ARRAY IP, THE SINGLE PRECISION ARRAYS PW, DY,             *
C* PRECISION ARRAYS SAVE AND CSAVE.  THE ARRAY PW IS USED ONLY TO HOLD *
C* THE MATRIX OF THE SAME NAME, AND SAVE IS USED TO SAVE THE VALUES   *
C* OF Y IN CASE A STEP HAS TO BE REPEATED, BUT CSAVE IS USED TO HOLD  *
C* SEVERAL ARRAYS.                                                   *
C*                                                                   *
C* THE PARAMETERS TO THE SUBROUTINE DIFSUB HAVE                      *
C* THE FOLLOWING MEANINGS..                                          *
C*                                                                   *
C*   N        THE NUMBER OF FIRST ORDER DIFFERENTIAL EQUATIONS.  N   *
C*              MAY BE DECREASED ON LATER CALLS IF THE NUMBER OF     *
C*              ACTIVE EQUATIONS REDUCES, BUT IT MUST NOT BE         *
C*              INCREASED WITHOUT CALLING WITH JSTART = 0.           *
C*   T        THE INDEPENDENT VARIABLE.                            *
C*   Y        A  7 BY N ARRAY CONTAINING THE DEPENDENT VARIABLES AND *
C*              THEIR SCALED DERIVATIVES.  Y(J+1,I) CONTAINS       *
C*              THE J-TH DERIVATIVE OF Y(I) SCALED BY              *
C*              H**J/FACTORIAL(J) WHERE H IS THE CURRENT           *
C*              STEP SIZE. ONLY Y(1,I) NEED BE PROVIDED BY         *
C*              THE CALLING PROGRAM ON THE FIRST ENTRY.            *
C*   H        THE STEP SIZE TO BE ATTEMPTED ON THE NEXT STEP.      *
C*              H MAY BE ADJUSTED UP OR DOWN BY THE PROGRAM        *
C*              IN ORDER TO ACHEIVE AN ECONOMICAL INTEGRATION.     *
C*              HOWEVER, IF THE H PROVIDED BY  *MAIN*  DOES        *
C*              NOT CAUSE A LARGER ERROR THAN REQUESTED, IT        *
C*              WILL BE USED.  TO SAVE COMPUTER TIME, *MAIN*       *
C*              USES A FAIRLY SMALL STEP FOR THE FIRST             *
C*              CALL.  IT WILL BE AUTOMATICALLY INCREASED LATER.   *
C*   HMIN     THE MINIMUM STEP SIZE THAT WILL BE USED.             *
C*   HMAX     THE MAXIMUM SIZE TO WHICH THE STEP WILL BE INCREASED *
C*   EPS      THE ERROR TEST CONSTANT.  SINGLE STEP ERROR ESTIMATES *
C*              DIVIDED BY  YMAX(I)  MUST BE LESS THAN THIS        *
C*              IN THE EUCLIDEAN NORM. THE STEP AND/OR ORDER IS    *
C*              ADJUSTED TO ACHEIVE THIS.                          *
C*   YMAX     AN ARRAY OF  N LOCATIONS WHICH CONTAINS THE MAXIMUM  *
C*              OF EACH Y SEEN SO FAR.  IT SHOULD NORMALLY BE SET TO *
C*              1 IN EACH COMPONENT BEFORE THE FIRST ENTRY. (SEE THE *
C*              DESCRIPTION OF EPS.)                               *
C*   ERROR    AN ARRAY OF  N ELEMENTS WHICH CONTAINS THE ESTIMATED *
C*              ONE STEP ERROR IN EACH COMPONENT.                  *
C*   KFLAG    A COMPLETION CODE WITH THE FOLLOWING MEANINGS..      *
C*                    +1  THE STEP WAS SUCCESFUL.                  *
C*                    -1  UNRECOVERABLE ERROR                      *
C*                     0  INTEGRATION COMPLETED                    *
C*   JSTART   AN INPUT INDICATOR WITH THE FOLLOWING MEANINGS..     *
C*                     0  PERFORM THE FIRST STEP.  THE FIRST STEP  *
C*                          MUST BE DONE WITH THIS VALUE OF JSTART *
C*                          SO THAT THE SUBROUTINE CAN INITIALIZE  *
C*                          ITSELF.                                *
C*                    +1  TAKE A NEW STEP CONTINUING FROM THE LAST. *
C*              JSTART IS SET TO NQ, THE CURRENT ORDER OF THE METHOD *
C*              AT EXIT.  NQ IS ALSO THE ORDER OF THE MAXIMUM      *
C*              DERIVATIVE AVAILABLE.                              *
C*   PW       A BLOCK OF AT LEAST N**2 FLOATING POINT LOCATIONS.   *
C*********************************************************************
      DIMENSION Y(N,7),YMAX(N),SAVE(N,7),ERROR(N),PW(1),DY(N)#
     1  CSAVE(N,2),IP(N)
      IRET = 1
      KFLAG = 1
      IF (JSTART.LE.0) GO TO 140
```

```
C*********************************************************************
C* BEGIN BY SAVING INFORMATION FOR POSSIBLE RESTARTS AND CHANGING    *
C* H BY THE FACTOR R IF THE CALLER HAS CHANGED H.  ALL VARIABLES     *
C* DEPENDENT ON H MUST ALSO BE CHANGED.                              *
C* E IS A COMPARISON FOR ERRORS OF THE CURRENT ORDER NQ. EUP IS      *
C* TO TEST FOR INCREASING THE ORDER, EDWN FOR DECREASING THE ORDER.  *
C* HNEW IS THE STEP SIZE THAT WAS USED ON THE LAST CALL.             *
C*********************************************************************
 100  DO 110 I = 1,N
        DO 110 J = 1,K
 110      SAVE(I,J) = Y(I,J)
      HOLD = HNEW
      IF (H.EQ.HOLD) GO TO 130
 120  IRET1 = 1
      GO TO 750
 130  NQOLD = NQ
      TOLD = T
      IF (JSTART.GT.0) GO TO 250
      GO TO 170
 140  IF (JSTART.EQ.-1) GO TO 160
C*********************************************************************
C* ON THE FIRST CALL, THE ORDER IS SET TO 1 AND THE INITIAL          *
C* DERIVATIVES ARE CALCULATED.                                       *
C*********************************************************************
      BR = 1.0
      NQ = 1
C     CALL DIFFUN(T,Y,DY)
      CALL LINK('DIFFUN')
      DO 150 I = 1,N
 150    Y(I,2) = DY(I)*H
      HNEW = H
      K = 2
      GO TO 100
C*********************************************************************
C* REPEAT LAST STEP BY RESTORING SAVED INFORMATION.                  *
C*********************************************************************
 160  IF (NQ.EQ.NQOLD) JSTART = 1
      T = TOLD
      NQ = NQOLD
      K = NQ + 1
      GO TO 120
C 170 CALL SUB1(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
 170  CALL LINK('SUB1')
C 230 CALL SUB2(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
 230  CALL LINK('SUB2')
      GO TO (240,780),JLNK
 240  GO TO ( 250 , 640 ),IRET
C 250 CALL SUB3(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
 250  CALL LINK('SUB3')
C*********************************************************************
C*    UP TO 2 CORRECTOR ITERATIONS ARE TAKEN.  CONVERGENCE IS TESTED BY *
C*    REQUIRING THE L2 NORM OF CHANGES TO BE LESS THAN BND WHICH IS  *
C*    DEPENDENT ON THE ERROR TEST CONSTANT.                          *
C*       THE SUM OF THE CORRECTIONS IS ACCUMULATED IN THE ARRAY      *
C*    ERROR(I).  IT IS EQUAL TO THE K-TH DERIVATIVE OF Y MULTIPLIED  *
C*    BY  H**K/(FACTORIAL(K-1)*A(K)), AND IS THEREFORE PROPORTIONAL  *
C*    TO THE ACTUAL ERRORS TO THE LOWEST POWER OF H PRESENT. (H**K)  *
C*********************************************************************
      DO 270 I = 1,N
 270    ERROR(I) = 0.0
      DO 430   L=1,2
C         CALL DIFFUN(T,Y,DY)
      CALL LINK('DIFFUN')
      DO 280  I=1,N
 280  CSAVE(I,1) = DY(I)
      JLNK = 1
      IF(IWEVAL.LT.1) JLNK = 2
      IF(JLNK.EQ.2) GO TO 350
C*********************************************************************
C*    IF THERE HAS BEEN A CHANGE OF ORDER OR THERE HAS BEEN TROUBLE  *
C*    WITH CONVERGENCE, PW IS RE-EVALUATED PRIOR TO STARTING THE     *
C*    CORRECTOR ITERATION IN THE CASE OF STIFF METHODS.  IWEVAL IS   *
C*    THEN SET TO -1 AS AN INDICATOR THAT IT HAS BEEN DONE.          *
C*********************************************************************
 310    DO 340 J = 1,N
      F = Y(J,1)
         R = EPS*AMAX1(EPS, ABS(F))
      Y(J,1) = Y(J,1) + R
         D = A(1)*H/R
C     CALL DIFFUN(T,Y,DY)
      CALL LINK('DIFFUN')
         DO 330 I = 1,N
 330       PW(I+(J-1)*N) = (DY(I)-CSAVE(I,1))*D
 340      Y(J,1) = F
```

```
C*********************************************************************
C* ADD THE IDENTITY MATRIX TO THE JACOBIAN AND DECOMPOSE INTO LU = PW  *
C*********************************************************************
 290     DO 300 I = 1,N
 300       PW(I*(N+1)-N) = 1.0 + PW(I*(N+1)-N)
         IWEVAL = -1
C 350 CALL SUB7(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
 350   CALL LINK('SUB7')
       GO TO (440,410),JLNK
C 410 CALL SUB4(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
 410   CALL LINK('SUB4')
       GO TO (430,440),JLNK
 430   CONTINUE
C 440 CALL SUB5(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
 440   CALL LINK('SUB5')
       GO TO (750,715,550,700,540),JLNK
C*********************************************************************
C* REDUCE THE FAILURE FLAG COUNT TO CHECK FOR MULTIPLE FAILURES.       *
C* RESTORE T TO ITS ORIGINAL VALUE AND TRY AGAIN UNLESS THERE HAVE     *
C* THREE FAILURES.  IN THAT CASE THE DERIVATIVES ARE ASSUMED TO HAVE   *
C* ACCUMULATED ERRORS SO A RESTART FROM THE CURRENT VALUES OF Y IS     *
C* TRIED. THIS IS CONTINUED UNTIL SUCCESS OR H = HMIN.                 *
C*********************************************************************
 540   KFLAG = KFLAG - 2
       IF(ABS(H).LE.HMIN) GO TO 740
       T = TOLD
       IF (KFLAG.LE.-5) GO TO 720
 550   JLNK = 2
C 640 CALL SUB6(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
 640   CALL LINK('SUB6')
       GO TO (700,750,170,250),JLNK
 700   DO 710 I = 1,N
 710     YMAX(I) = AMAX1(YMAX(I),ABS(Y(I,1)))
       JSTART = NQ
 715   RETURN
 720   IF(NQ.GT.1) GO TO 725
       IF( ABS(H).LE.2.E0*HMIN) GO TO 700
       GO TO 550
 725   R = H/HOLD
       DO 730   I = 1,N
       Y(I,1) = SAVE(I,1)
 730     Y(I,2) = SAVE(I,2)*R
       NQ = 1
       KFLAG = 1
       GO TO 170
 740   KFLAG = -1
       HNEW = H
       JSTART = NQ
       RETURN
C 750 CALL SUB8(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
 750   CALL LINK('SUB8')
       GO TO (130,250,640,440),JLNK
 700    KFLAG = -4
       RETURN
       END
```

Sample user-provided Overlay; Y" = -Y

```
.TITLE INTRO
.GLOBL N,RETURN,DIFFUN
.CSECT ADDR
ADDR: .=.
.CSECT COM
T=.+74
.CSECT
BEGIN:    MOV ADDR+10,LIST+2
          MOV ADDR,LIST+4
          JSR %5,DIFFUN
          BR RET
LIST:     .WORD T,0,0
RET:      JSR %5,RETURN
.END BEGIN
```

```
      SUBROUTINE DIFFUN(T,Y,DY)
      DIMENSION Y(2,7), DY(2)
      DY(1) = Y(2,1)
      DY(2) = -Y(1,1)
      RETURN
      END
```

Regular Overlays; INTRO with the appropriate subroutine name introduces each overlay

```
.TITLE INTRO
.GLOBL N,RETURN,SUB1
.CSECT ADDR
ADDR: .=.
.CSECT
BEGIN:    MOV ADDR+10,LIST+2
          MOV ADDR+12,LIST+4
          MOV ADDR+14,LIST+6
          MOV ADDR+4,LIST+10
          MOV ADDR+2,LIST+12
          MOV ADDR+16,LIST+14
          MOV ADDR+6,LIST+16
          JSR %5,SUB1
          BR RET
LIST:     .WORD N,0,0,0,0,0,0,0
RET:      JSR %5,RETURN
.END BEGIN
```

```
      SUBROUTINE SUB1(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
      DIMENSION Y(N,7),SAVE(N,7),CSAVE(N,2),YMAX(N),
     1          ERROR(N),PW(N,N),IP(N)
C*************************************************************************
C* SET THE COEFFICIENTS THAT DETERMINE THE ORDER AND THE METHOD          *
C* TYPE.  CHECK FOR EXCESSIVE ORDER.  THE LAST TWO STATEMENTS OF         *
C* THIS SECTION  SET IWEVAL .GT.0 IF PW IS TO BE RE-EVALUATED            *
C* BECAUSE OF THE ORDER CHANGE, AND THEN REPEAT THE INTEGRATION          *
C* STEP IF IT WAS NOT YET BEEN DONE (IRET = 1) OR SKIP TO A FINAL        *
C* SCALING BEFORE EXIT IF IT WAS BEEN COMPLETED (IRET = 2).              *
C*************************************************************************
      A(2) = -1.0
      IF (NQ .GT. 6) NQ=6
      GO TO (221,222,223,224,225,226),NQ
C*************************************************************************
C* THE FOLLOWING COEFFICIENTS SHOULD BE DEFINED TO THE MAXIMUM           *
C* ACCURACY PERMITTED BY THE MACHINE.  THEY ARE, IN THE ORDER USED..     *
C*                                                                       *
C*                                                                       *
C* -1                                                                    *
C* -2/3,-1/3                                                             *
C* -12/25,-7/10,-1/5,-1/50                                               *
C* -120/274,-225/274,-85/274,-15/274,-1/274                              *
C* -180/441,-58/63,-15/36,-25/252,-3/252,-1/1764                         *
C*************************************************************************
 221  A(1) = -1.000000000
      GO TO 230
 222  A(1) = -0.6666666666666667
      A(3) = -0.3333333333333333
      GO TO 230
 223  A(1) = - 0.5454545454545455
      A(3) = A(1)
      A(4) = -0.09090909090909091
      GO TO 230
 224  A(1) = -0.480000000
      A(3) = -0.700000000
      A(4) = -0.200000000
      A(5) = -0.020000000
      GO TO 230
 225  A(1) = -0.437956204379562
      A(3) = -0.8211678832116788
      A(4) = -0.3102189781021898
      A(5) = -0.05474452554744526
      A(6) = -0.003649635036496350
      GO TO 230
 226  A(1) = -0.4081632653061225
      A(3) = -0.9206349206349206
      A(4) = -0.4166666666666667
      A(5) = -0.0992063492063492
      A(6) = -0.01190476190476190
      A(7) = -0.0005668934240362812
  230  RETURN
      END
```

```
      SUBROUTINE SUB2(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
      DIMENSION Y(N,7),SAVE(N,7),CSAVE(N,2),YMAX(N),
     1         ERROR(N),PW(N,N),IP(N)
      DIMENSION PERTST(7,3)
C*********************************************************************
C* THE COEFFICIENTS IN PERTST ARE USED IN SELECTING THE STEP AND     *
C* ORDER, THEREFORE ONLY ABOUT ONE PERCENT ACCURACY IS NEEDED.       *
C*********************************************************************
      DATA PERTST/2.,4.5,7.333,10.42,13.7,17.15,1.,
     1     3.,6.,9.167,12.5,15.98,1.,1.,
     2      1.,1.,.5,.1667,.04167,.008333/
      JLNK = 1
  230 K = NQ+1
      IDOUB = K
      ENQ2 = .5/FLOAT(NQ + 1)
      ENQ3 = .5/FLOAT(NQ + 2)
      ENQ1 = 0.5/FLOAT(NQ)
      EUP = (PERTST(NQ,2)*EPS)**2
      E = (PERTST(NQ,1)*EPS)**2
      EDWN = (PERTST(NQ,3)*EPS)**2
      IF(EDWN.EQ.0) JLNK = 2
      BND = (EPS*ENQ3)**2
  240 IWEVAL = 2
      RETURN
      END



      SUBROUTINE SUB3(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
      DIMENSION Y(N,7),SAVE(N,7),CSAVE(N,2),YMAX(N),
     1         ERROR(N),PW(N,N),IP(N)
C*********************************************************************
C* THIS SECTION COMPUTES THE PREDICTED VALUES BY EFFECTIVELY          *
C* MULTIPLYING THE SAVED INFORMATION BY THE PASCAL TRIANGLE           *
C* MATRIX.                                                           *
C*********************************************************************
  250 T = T + H
      DO 260 J = 2,K
        DO 260 J1 = J,K
          J2 = K - J1 + J - 1
          DO 260 I = 1,N
  260       Y(I,J2) = Y(I,J2) + Y(I,J2+1)
      RETURN
      END



      SUBROUTINE SUB4(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
      DIMENSION Y(N,7),SAVE(N,7),CSAVE(N,2),YMAX(N),
     1         ERROR(N),PW(N,N),IP(N)
C*********************************************************************
C* CORRECT AND COMPARE DEL, THE L2 NORM OF CHANGE/YMAX, WITH BND.    *
C* ESTIMATE THE VALUE OF THE L2 NORM OF THE NEXT CORRECTION BY       *
C* BR*2*DEL AND COMPARE WITH BND.  IF EITHER IS LESS, THE CORRECTOR  *
C* IS SAID TO HAVE CONVERGED.                                        *
C*********************************************************************
  410 DEL = 0.0E0
      DO 420 I = 1,N
        Y(I,1) = Y(I,1) + A(1)*CSAVE(I,1)
        Y(I,2) = Y(I,2) - CSAVE(I,1)
        ERROR(I) = ERROR(I) + CSAVE(I,1)
      DEL = DEL + (CSAVE(I,1)/YMAX(I))**2
  420 CONTINUE
  426 IF(L.GE.2) BR = AMAX1(.9*BR, DEL/DEL1)
      DEL1 = DEL
      JLNK = 1
      IF(AMIN1(DEL,BR*DEL*2.0)  .LE.  BND)  JLNK = 2
      RETURN
      END
```

```
      SUBROUTINE SUB5(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
      DIMENSION Y(N,7),SAVE(N,7),CSAVE(N,2),YMAX(N),
     1          ERROR(N),PW(N,N),IP(N)
C*********************************************************************
C* THE CORRECTOR ITERATION FAILED TO CONVERGE IN 2 TRIES.  VARIOUS   *
C* POSSIBILITIES ARE CHECKED FOR.  IF H IS ALREADY HMIN AND          *
C* THIS IS EITHER ADAMS METHOD OR THE STIFF METHOD IN WHICH THE      *
C* MATRIX PW HAS ALREADY BEEN RE-EVALUATED, A NO CONVERGENCE EXIT    *
C* IS TAKEN. OTHERWISE THE MATRIX PW IS RE-EVALUATED AND/OR THE      *
C* STEP IS REDUCED TO TRY AND GET CONVERGENCE.                       *
C*********************************************************************
      GO TO (440,490,470,470),JLNK
 440  T = TOLD
      IF(ABS(H).LE.HMIN .AND. (IWEVAL-1).LT.-1) GO TO 460
      IF(IWEVAL.NE.0) H = H*0.25E0
      IWEVAL = 2
      IRET1 = 2
      JLNK = 1
      GO TO 536
 460  KFLAG = -3
      NQ = NQOLD
 470  DO 480 I = 1,N
        DO 480 J = 1,K
 480      Y(I,J) = SAVE(I,J)
      H = HOLD
      JSTART = NQ
      JLNK = 2
      GO TO 536
C*********************************************************************
C* THE CORRECTOR CONVERGED AND CONTROL IS PASSED TO STATEMENT 520    *
C* IF THE ERROR TEST IS O.K.,  AND TO 540 OTHERWISE.                 *
C* IF THE STEP IS O.K. IT IS ACCEPTED. IF IDOUB HAS BEEN REDUCED     *
C* TO ONE,  A TEST IS MADE TO SEE IF THE STEP CAN BE INCREASED       *
C* AT THE CURRENT ORDER OR BY GOING TO ONE HIGHER OR ONE LOWER.      *
C* SUCH A CHANGE IS ONLY MADE IF THE STEP CAN BE INCREASED BY AT     *
C* LEAST 1.1.  IF NO CHANGE IS POSSIBLE IDOUB IS SET TO 8 TO         *
C* PREVENT FUTHER TESTING FOR 8 STEPS.                               *
C* IF A CHANGE IS POSSIBLE, IT IS MADE AND IDOUB IS SET TO           *
C* NQ + 1     TO PREVENT FURTHER TESING FOR THAT NUMBER OF STEPS.    *
C* IF THE ERROR WAS TOO LARGE, THE OPTIMUM STEP SIZE FOR THIS OR     *
C* LOWER ORDER IS COMPUTED, AND THE STEP RETRIED.  IF IT SHOULD      *
C* FAIL TWICE MORE IT IS AN INDICATION THAT THE DERIVATIVES THAT     *
C* HAVE ACCUMULATED IN THE Y ARRAY HAVE ERRORS OF THE WRONG ORDER    *
C* SO THE FIRST DERIVATIVES ARE RECOMPUTED AND THE ORDER IS SET      *
C* TO 1.                                                             *
C*********************************************************************
 490  D = 0.0
      DO 500 I = 1,N
 500    D = D + (ERROR(I)/YMAX(I))**2
      IWEVAL = 0
      IF(D.GT.E) JLNK = 5
      IF(JLNK.EQ.5) GO TO 536
      IF (K.LT.3) GO TO 520
C*********************************************************************
C* COMPLETE THE CORRECTION OF THE HIGHER ORDER DERIVATIVES AFTER A    *
C* SUCCESFUL STEP.                                                   *
C*********************************************************************
      DO 510 J = 3,K
        DO 510 I = 1,N
 510      Y(I,J) = Y(I,J) + A(J)*ERROR(I)
 520  KFLAG = +1
      HNEW = H
      IF(IDOUB.LE.1) JLNK=3
      IF(JLNK.EQ.3) GO TO 536
      IDOUB = IDOUB - 1
      IF(IDOUB.GT.1) GO TO 535
      DO 530 I = 1,N
 530    CSAVE(I,2) = ERROR(I)
 535  JLNK = 4
 536  RETURN
      END
```

```
      SUBROUTINE SUB6(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
      DIMENSION Y(N,7),SAVE(N,7),CSAVE(N,2),YMAX(N),
     1        ERROR(N),PW(N,N),IP(N)
C*************************************************************************
C* PR1, PR2, AND PR3  WILL CONTAIN THE AMOUNTS BY WHICH THE STEP SIZE    *
C* CHOULD BE DIVIDED AT ORDER ONE LOWER, AT THIS ORDER, AND AT ORDER     *
C* ONE HIGHER RESPECTIVELY.                                              *
C*************************************************************************
      GO TO (680,550,640),JLNK
 550  PR2 = (D/E)**ENQ2*1.2
      PR3 = 1.E+20
      IF((NQ.GE.6).OR.(KFLAG.LE.-1))GO TO 570
      D = 0.0
      DO 560 I = 1,N
 560    D = D + ((ERROR(I) - CSAVE(I,2))/YMAX(I))**2
      PR3 = (D/EUP)**ENQ3*1.4
 570  PR1 = 1.E+20
      IF (NQ.LE.1) GO TO 590
      D = 0.0
      DO 580 I = 1,N
 580    D = D + (Y(I,K)/YMAX(I))**2
      PR1 = (D/EDWN)**ENQ1*1.3
 590  CONTINUE
      IF (PR2.LE.PR3) GO TO 650
      IF (PR3.LT.PR1) GO TO 660
 600  R = 1.0/AMAX1(PR1,1.E-4)
      NEWQ = NQ - 1
  610 IDOUB = 8
      IF ((KFLAG.EQ.1).AND.(R.LT.(1.1))) GO TO 694
      IF (NEWQ.LE.NQ) GO TO 630
C*************************************************************************
C* COMPUTE ONE ADDITIONAL SCALED DERIVATIVE IF ORDER IS INCREASED.       *
C*************************************************************************
      DO 620 I = 1,N
  620   Y(I,NEWQ+1) = ERROR(I)*A(K)/FLOAT(K)
 630  K = NEWQ + 1
      IF (KFLAG.EQ.1) GO TO 670
      H = H*R
      IRET1 = 3
      JLNK = 2
      GO TO 695
 640  IF (NEWQ.EQ.NQ) JLNK=4
      IF(JLNK.EQ.4) GO TO 695
      NQ = NEWQ
      JLNK = 3
      GO TO 695
 650  IF (PR2.GT.PR1) GO TO 600
      NEWQ = NQ
      R = 1.0/AMAX1(PR2,1.E-4)
      GO TO 610
 660  R = 1.0/AMAX1(PR3,1.E-4)
      NEWQ = NQ + 1
      GO TO 610
 670  IRET = 2
      R = AMIN1(R,HMAX/ABS(H))
      H = H*R
      HNEW = H
      IF (NQ.EQ.NEWQ) GO TO 680
      NQ = NEWQ
      JLNK = 3
      GO TO 695
 680  R1 = 1.0
      DO 690 J = 2,K
        R1 = R1*R
        DO 690 I = 1,N
 690      Y(I,J) = Y(I,J)*R1
      IDOUB = K
 694  JLNK = 1
 695   RETURN
      END
```

```
      SUBROUTINE SUB7(N,Y,SAVE,B,YMAX,ERROR,A,IP)
      COMMON /COM/ X(7),JLNK,JSTART,KK,KFLAG,NQ,NQOLD,NEWQ,EPS,XT,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
C
C     MATRIX TRIANGULARIZATION BY GAUSSIAN ELIMINATION.
C
C    INPUT...
C      N = ORDER OF MATRIX
C      NDIM = DECLARED DIMENSION OF ARRAY A.
C      A = MATRIX TO BE TRIANGULARIZED. (FOR STIFF METHODS, A IS SINGLE
C                  PRECISION;  ALL OTHER VARIABLE ARE DOUBLE PRECISION.)
C    OUTPUT...
C      A(I,J), I.LE.J  = UPPER TRIANGULAR FACTOR, U.
C      A(I,J),   I.GT.J   = MULTIPLIERS  = LOWER TRIANGULAR FACTOR, I-L.
C      IP(K),  K.LT.N = INDEX OF K-TH PIVOT ROW.
C      IP(N)  = (-1)**(NUMBER OF INTERCHANGES) OR 0.
C    USE 'SOLVE' TO OBTAIN SOLUTION OF LINEAR SYSTEM.
C    DETERM(A) = IP(N)*A(1,1)*A(2,2)*...*A(N,N).
C    IF IP(N) = 0, A IS SINGULAR, SOLVE WILL DIVIDE BY ZERO.
C
      DIMENSION A(N,N),B(1),IP(N),Y(1)
      GO TO (101,350),JLNK
  101 JLNK = 2
      NDIM = N
      IP(N) = 1
      DO 6  K=1,N
      IF(K.EQ.N) GO TO 5
      KP1 = K+1
      M = K
      DO 1  I=KP1,N
      IF(ABS(A(I,K)).GT.ABS(A(M,K)))   M=I
    1 CONTINUE
      IP(K) = M
      IF(M.NE.K)  IP(N) = -IP(N)
      T = A(M,K)
      A(M,K) = A(K,K)
      A(K,K) = T
      IF(T.EQ.0) GO TO 5
      DO 2  I=KP1,N
    2    A(I,K) = -A(I,K) / T
      DO 4  J=KP1,N
      T = A(M,J)
      A(M,J) = A(K,J)
      A(K,J) = T
      IF(T.EQ.0.) GO TO 4
      DO 3   I=KP1,N
    3    A(I,J) = A(I,J) + A(I,K)*T
    4  CONTINUE
    5 IF(A(K,K).EQ.0.) IP(N) = 0
    6 CONTINUE
      IF(IP(N).EQ.0) JLNK = 1
      IF(JLNK.EQ.1) GO TO 10
  350 DO 360 I=1,N
  360  B(I) = Y(I+N) - B(I)*H
      JLNK = 2
C
C   SOLUTION OF LINEAR SYSTEM, A*X = B.
C
C   INPUT...
C     N = ORDER OF MATRIX.
C     NDIM = DECLARED DIMENSION OF ARRAY A.
C     A = TRIANGULARIZED MATRIX OBTAINED FROM 'DECOMP'.
C     B = RIGHT HAND SIDE VECTOR.
C     IP = PIVOT VECTOR OBTAINED FROM 'DECOMP'.
C   OUTPUT...
C     B = SOLUTION VECTOR, X.
      IF(N.EQ.1) GO TO 9
      NM1 = N-1
      DO 7  K=1,NM1
         KP1 = K + 1
         M = IP(K)
      T = B(M)
      B(M) = B(K)
      B(K) = T
      DO 7  I=KP1,N
    7 B(I) = B(I) + A(I,K)*T
      DO 8   KB=1,NM1
      KM1 = N - KB
      K = KM1 + 1
      B(K) = B(K)/A(K,K)
      T = -B(K)
      DO 8  I=1,KM1
    8 B(I) = B(I) + A(I,K)*T
    9 B(1) = B(1)/A(1,1)
   10 RETURN
      END
```

```
      SUBROUTINE SUB0(N,Y,SAVE,CSAVE,YMAX,ERROR,PW,IP)
      COMMON /COM/ A(7),JLNK,JSTART,K,KFLAG,NQ,NQOLD,NEWQ,EPS,T,TOLD,
     1  IDOUB,IWEVAL,IRET,IRET1,H,HOLD,HNEW,HMAX,HMIN,E,EUP,EDWN,ENQ1,
     2ENQ2,ENQ3,BND,BR,DEL,DEL1,D
      DIMENSION Y(N,7),SAVE(N,7),CSAVE(N,2),YMAX(N),
     1         ERROR(N),PW(N,N),IP(N)
C*********************************************************************
C* THIS SECTION SCALES ALL VARIABLES CONNECTED WITH H AND RETURNS     *
C* TO THE ENTERING SECTION.                                          *
C*********************************************************************
  750   H = AMAX1(HMIN,AMIN1(H,HMAX))
      R1 = 1.0
      DO 760 J = 2,K
      R = H/HOLD
      R1 = R1*R
        DO 760 I = 1,N
  760     Y(I,J) = SAVE(I,J)*R1
      DO 770 I = 1,N
  770   Y(I,1) = SAVE(I,1)
      IDOUB = K
      JLNK = IRET1
      GO TO 785
  780 KFLAG = -4
      JLNK = 4
  785  RETURN
      END
```

Form AEC–427
(6/68)
AECM 3201

U.S. ATOMIC ENERGY COMMISSION
UNIVERSITY–TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO.
COO-1469-0215

2. TITLE
NUMERICAL SYSTEMS ON A MINICOMPUTER

Roy Leonard Brown, Jr.

3. TYPE OF DOCUMENT (Check one):

- [X] a. Scientific and technical report
- [ ] b. Conference paper not to be published in a journal:
  - Title of conference _____
  - Date of conference _____
  - Exact location of conference _____
  - Sponsoring organization _____
- [ ] c. Other (Specify) _____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- [X] a. AEC's normal announcement and distribution procedures may be followed.
- [ ] b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
- [ ] c. Make no announcement or distrubution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

C. W. Gear
Professor and Principal Investigator

Organization

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Signature Charles W Gear.

Date February 1973

FOR AEC USE ONLY

AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

PATENT CLEARANCE:

- [ ] a. AEC patent clearance has been granted by responsible AEC patent group.
- [ ] b. Report has been sent to responsible AEC patent group for clearance.
- [ ] c. Patent clearance not required.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-73-555	2.	3. Recipient's Accession No.
4. Title and Subtitle NUMERICAL SYSTEMS ON A MINICOMPUTER			5. Report Date February 1973 6.
7. Author(s) Roy Leonard Brown, Jr.			8. Performing Organization Rept. No. COO-1469-0215
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, Illinois 61801			10. Project/Task/Work Unit No. US AEC AT(11-1)1469 11. Contract/Grant No.
12. Sponsoring Organization Name and Address US AEC Chicago Operations Office 9800 South Cass Avenue Argonne, Illinois 60439			13. Type of Report & Period Covered Thesis Research 14.

15. Supplementary Notes

16. Abstracts

This thesis defines the concept of a numerical system for a minicomputer and provides a description of the software and computer system configuration necessary to implement such a system. A procedure for creating a numerical system from a FORTRAN program is developed and an example is presented. The reader should have some knowledge of FORTRAN and minicomputer operating systems, PAL assembly language for PDP-11.

17. Key Words and Document Analysis. 17a. Descriptors

minicomputer system
numerical system
overlay buffer
overlay partition of FORTRAN program
overlay facility

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
unlimited distribution	UNCLASSIFIED	48
	20. Security Class (This Page) UNCLASSIFIED	22. Price